COMMENTS

C1: Inappropriate Information

Comments are written only when needed to be written, comments about the functionality of the function are included in the documentation of the function.

C2: Obsolete Comment

Comments that are obsolete were removed.

C3: Redundant Comment

Comments are written only when needed to be written.

C4: Poorly Written Comment

I use proper english language to explain the code.

C5: Commented-Out Code

There is no commented-out code.

ENVIRONMENT

E1: Build Requires More Than One Step

I built the project using Maven, which is a one command to run.

FUNCTIONS

F1: Too Many Arguments

Most functions don't have many arguments.

F2: Output Arguments

All arguments used as inputs.

F3: Flag Arguments

There are no flag arguments in my methods.

F4: Dead Function

All functions are used.

GENERAL

G1: Multiple Languages in One Source File

All my code is written in java.

G2: Obvious Behavior Is Unimplemented

The code runs as you would expect it to be.

G3: Incorrect Behavior at the Boundaries

I tested out the corner cases manually.

G4: Overridden Safeties

All warnings by the compiler were taken into account.

G5: Duplication

I make sure I don't repeat the code.

G6: Code at Wrong Level of Abstraction

In Request , ByIdRequest, EntityByIdRequest and EntityRequest, I made sure all the functionality that could be moved to the higher abstraction are in Request Class.

G7: Base Classes Depending on Their Derivatives

Request is completely separate of ByIdRequest, EntityByIdRequest and EntityRequest and doesn't depend on it in any way.

G8: Too Much Information

I used the simplest GUI I could come up with to run the application.

G9: Dead Code

I made sure to add try-catch statements and if statements only when necessary.

G10: Vertical Separation

I tried to declare local variables in the smallest scope right at their first usage.

G11: Inconsistency

There is no inconsistency in the behavior or the design.

G12: Clutter

There is no useless code.

G13: Artificial Coupling

There is no artificial coupling in the code.

G14: Feature Envy

The code was placed where it would feel natural.

G16: Obscured Intent

There is no obscured intent on the code.

G17: Misplaced Responsibility

The code was placed where it would feel natural.

G18: Inappropriate Static

I used static methods where the data wasn't related to a specific object.

G19: Use Explanatory Variables

I wrote the code to be expressive so it would be easy to understand the intention of it, by using meaningful classes, methods and variables names, and splitting complicated code to many functions that do single functionality.

G20: Function Names Should Say What They Do

Function names describe what you would expect from them to do.

G21: Understand the Algorithm

I wrote the code understanding exactly what should happen.

G22: Make Logical Dependencies Physical

Most of the dependencies are injected to their dependents.

G23: Prefer Polymorphism to If/Else or Switch/Case

My code is designed for posterity using interfaces so that we can migrate to the next functionality using Polymorphism.

G24: Follow Standard Conventions

My Code follows the standard conventions of Java as explained in the report if "Google styling guide, Java Effective Code"

G25: Replace Magic Numbers with Named Constants

All numbers used were changed to constants such as the server (port number), Connection pool (initial size and max size) and Cache (max size).

G26: Be Precise

All decisions were all well thought out and planned so they fit the best practices.

G28: Encapsulate Conditionals

All of the conditions are clear to the reader.

G29: Avoid Negative Conditionals

There is no ambiguity in conditions.

G30: Functions Should Do One Thing

Functions are written to do only one task.

G32: Don’t Be Arbitrary

Functions are written to do only one task.

G33: Encapsulate Boundary Conditions

There is no obscured intent on the code.

G34: Functions Should Descend Only One Level of Abstraction

There is no obscured intent on the code.

G35: Keep Configurable Data at High Levels

All Configurable data and constants are in a utility classes

G36: Avoid Transitive Navigation

Transitive navigation was minimized by functions that do only one thing.

JAVA

J1: Avoid Long Import Lists by Using Wildcards

I used wildcards when importing a list of constants.

J2: Don’t Inherit Constants

I don’t inherit constants.

J3: Constants versus Enums

I used enums to represent request type instead of using constants.

NAMES

N1: Choose Descriptive Names

All of the Classes, Variables and Methods have descriptive names.

N2: Choose Names at the Appropriate Level of Abstraction

All of the Classes have the correct name at the appropriate level of abstraction such as :

- SocketReader and StringSocketReader
- SocketWriter and StringSocketWriter
- ICache and EmployeeCache
- ConnectionPool and BasicConnectionPool
- DataManager and AppDataManager
- DBHelper and AppDBHelper

N3: Use Standard Nomenclature Where Possible

All of the Classes, Variables and Methods have a standard nomenclature.

N4: Unambiguous Names

All of the names are clear to the reader.

N6: Avoid Encodings

Explained in N4.

N7: Names Should Describe Side-Effects

All of the names are describe side-effects such as:

- getAndPrintResponse() method that get the response and then prints it.